

05/24/99

A

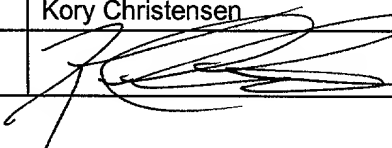
PTO/SB/05 (2/98) (modified)

Approved for use through 9/30/2000, OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

<b>NEW UTILITY PATENT APPLICATION TRANSMITTAL</b> <i>(only for new nonprovisional applications under 37 CFR 1.53(b))</i>	Attorney Docket Number	4278
	First Named Inventor	Peter B. Kessler, et al.
	Total Pages in this Submission	11
	Express Mail Label No.	EL021106727US

05/24/99

APPLICATION ELEMENTS	ACCOMPANYING APPLICATION PARTS
1. <input checked="" type="checkbox"/> Fee Transmittal Form (in duplicate) <input checked="" type="checkbox"/> Check Enclosed 2. <input checked="" type="checkbox"/> Specification <i>(preferred arrangement set forth below)</i> <input type="checkbox"/> Descriptive Title of the Invention <input type="checkbox"/> Cross Reference(s) to Related Case(s) <input type="checkbox"/> Statement Regarding Fed sponsored R & D <input type="checkbox"/> Background of the Invention <input type="checkbox"/> Brief Summary of the Invention <input type="checkbox"/> Brief Description of the Drawing(s) <input type="checkbox"/> Detailed Description <input type="checkbox"/> Claim or Claims <input type="checkbox"/> Abstract of the Disclosure 3. <input checked="" type="checkbox"/> Drawing(s) <i>( when necessary per 35 USC 113)</i> 4. Oath or Declaration a. <input type="checkbox"/> New Declaration <input type="checkbox"/> Executed b. <input checked="" type="checkbox"/> Copy from a prior application (37 CFR 1.63(d)) <i>(for continuation/divisional with Box 17 completed)</i> i. <input type="checkbox"/> DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b). 5. <input type="checkbox"/> Incorporation by Reference <i>(useable if Box 4b is checked). The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.</i>	6. <input type="checkbox"/> Assignment & Assignment Recordation Cover Sheet 7. <input type="checkbox"/> Certified Copy of Priority Document(s) <i>(if foreign priority is claimed)</i> 8. <input type="checkbox"/> Information Disclosure Statement & PTO-1449 <input type="checkbox"/> Copies of IDS Citation(s) 9. <input type="checkbox"/> Preliminary Amendment 10. Small Entity Statement <input type="checkbox"/> New Statement enclosed <input type="checkbox"/> Statement filed in prior application. Status still proper and desired 11. <input checked="" type="checkbox"/> Return Postcard 12. <input type="checkbox"/> 13. <input type="checkbox"/> 14. <input type="checkbox"/> 15. <input type="checkbox"/> 16. <input type="checkbox"/>
<b>ADDRESS TO:</b> <b>Assistant Commissioner for Patents</b> <b>Box Patent Application</b> <b>Washington, D.C. 20231</b>	
17. If a <b>CONTINUING APPLICATION</b> , check appropriate box and supply the requisite information below and in a preliminary amendment: <input checked="" type="checkbox"/> Continuation <input type="checkbox"/> Divisional <input type="checkbox"/> Continuation-in-part (CIP) of prior application No: <u>08/708,945</u> Prior application information:    Examiner: <u>P. Caldwell</u> Group/Art Unit: <u>2755</u>	
<b>18. CORRESPONDENCE ADDRESS</b>	
NAME	Kory Christensen    Fenwick & West LLP
ADDRESS	Two Palo Alto Square
CITY	Palo Alto    STATE CA    ZIP CODE 94306
COUNTRY	U.S.A.    TELEPHONE (650) 858-7657    FAX (650) 494-1417
Name (Print/Type)	Kory Christensen    Registration No. (Attorney/Agent) 43,548
Signature	 Date May 24, 1999

Rev. 05/24/99

IN THE UNITED STATES  
PATENT AND TRADEMARK OFFICE

APPLICANT(S): Peter B. Kessler, et al.  
SERIAL NO.: Unassigned  
FILING DATE: May 24, 1999  
TITLE: METHOD AND SYSTEM FOR TYPE IDENTIFICATION FOR MULTIPLE  
OBJECT INTERFACES IN A DISTRIBUTED OBJECT ENVIRONMENT  
EXAMINER: Unassigned  
GROUP ART UNIT: Unassigned  
ATTY. DKT. NO.: 4278 US (2450 US on parent)

---

**CERTIFICATE OF MAILING**

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Assistant Commissioner For Patents, Washington, D.C. 20231, on the date printed below:

Dated: May 24, 1999

By: 

Kory D. Christensen, Reg. No. 43,548

ASSISTANT COMMISSIONER FOR PATENTS  
WASHINGTON, D.C. 20231

PRELIMINARY AMENDMENT

Sir:

Prior to the examination of the patent application identified above, please amend  
the application as follows:

IN THE SPECIFICATION:

Please delete the first sentence on page 1 and insert the following:

--RELATED APPLICATIONS

This application is a continuation of U.S. Patent Application Serial No. 08/708,945, filed September 6, 1996, now \_\_\_\_\_, which is a divisional of U.S. Patent Application Serial No. 08/410,357, filed March 24, 1995, now abandoned.--

IN THE CLAIMS:

Please cancel claims 12-16 and 23-43 without prejudice.

Please add new claims 44-60 as follows:

1           44. (New) A computer system that resolves name collisions by providing  
2 type support for multiple type definitions, comprising:  
3           an interface repository including:  
4                 a repository naming context; and,  
5                 a prefix naming context subordinate to the repository naming  
6                         context, the prefix naming context serving as a root naming  
7                         context for at least one interface definition language  
8                         declaration, the prefix naming context being adapted to re-  
9                         solve names subordinate to the repository naming context.

1           45. (New) The system of claim 44 wherein the prefix naming context  
2 further includes:  
3           at least one naming context defined by an interface definition object and  
4                 subordinate to the prefix naming context.

1           46. (New) The computer system of claim 45 wherein at least one  
2 interface definition object has a fully scoped object name including a prefix name  
3 of the prefix naming context to which the interface definition object is  
4 subordinated.

1           47. (New) The computer system of claim 44 wherein the prefix naming  
2 context is immediately subordinate to the repository naming context.

1           48. (New) The computer system of claim 44 wherein the prefix naming  
2 context further includes:

3           at least one leaf node defined by an interface definition object.

1           49. (New) The computer system of claim 44, wherein the prefix naming  
2 context is defined by a prefix object.

1           50. (New) The computer system of claim 44, further comprising:  
2 an interface repository loader that accepts as input parameters a specified  
3 interface definition language file containing at least one interface  
4 definition language declaration, and a specified prefix name, and  
5 installs the at least one interface definition language declaration in  
6 a prefix naming context having the prefix naming context in the  
7 interface repository.

1           51. (New) A computer system that resolves name collisions by providing  
2 type support for multiple type definitions, comprising:

3           an interface repository including:

4           a repository naming context; and

5 a prefix naming context subordinate to the repository naming  
6 context, the prefix naming context serving as a root naming  
7 context for at least one interface definition language  
8 declaration, the prefix naming context being adapted to re-  
9 solve names subordinate to the repository naming context;  
10 and

11 an interface repository loader that accepts as input parameters a specified  
12 interface definition language file containing at least one interface  
13 definition language declaration, and a specified prefix name, and  
14 installs the at least one interface definition language declaration in  
15 a prefix naming context having the prefix naming context in the  
16 interface repository, and wherein the interface repository loader  
17 creates a data file identified as related to the specified interface  
18 definition language file, and containing an identification of the  
19 specified prefix naming context.

1 52. (New) A computer system that resolves name collisions by providing  
2 type support for multiple type definitions, comprising:

3 an interface repository including:

4 a repository naming context; and

5 a prefix naming context subordinate to the repository naming  
6 context, the prefix naming context serving as a root naming  
7 context for at least one interface definition language  
8 declaration, the prefix naming context being adapted to re-  
9 solve names subordinate to the repository naming context;  
10 and

11 an interface repository loader that accepts as input parameters a specified  
12 interface definition language file containing at least one interface  
13 definition language declaration, and a specified prefix name, and  
14 installs the at least one interface definition language declaration in  
15 a prefix naming context having the prefix naming context in the  
16 interface repository, and wherein the interface repository loader  
17 creates the specified prefix naming context in the interface  
18 repository if the specified prefix naming context does not exist  
19 therein.

1 53. (New) The computer system of claim 52, further comprising:  
2 a memory device that stores the interface repository; and  
3 a processing unit that executes operations of the interface repository  
4 loader.

1 54. (New) A computer system that resolves name collisions by providing  
2 type support for multiple type definitions, comprising:  
3 an interface repository including:  
4 a repository naming context; and  
5 a prefix naming context subordinate to the repository naming  
6 context, the prefix naming context serving as a root naming  
7 context for at least one interface definition language  
8 declaration, the prefix naming context being adapted to re-  
9 solve names subordinate to the repository naming context;  
10 an interface repository loader that accepts as input parameters a specified  
11 interface definition language file containing at least one interface  
12 definition language declaration, and a specified prefix name, and

13 installs the at least one interface definition language declaration in  
14 a prefix naming context having the prefix naming context in the  
15 interface repository;  
16 a memory device that stores the interface repository; and  
17 a processing unit that executes operations of the interface repository  
18 loader, and further executes the interface repository loader to  
19 create a data file identified as related to the specified interface  
20 definition language file, and containing an identification of the  
21 specified prefix naming context.

1 55. (New). A method of resolving name collisions by providing type  
2 support for multiple type definitions, comprising the steps of:  
3 defining in an interface repository a prefix naming context, the prefix  
4 naming context being adapted to resolve names subordinate to the  
5 repository naming context; and  
6 storing the prefix naming context subordinate to the repository naming  
7 context in the interface repository, the prefix naming context  
8 forming an interface definition language root context for interface  
9 definition objects subordinate to the prefix naming context.

1 56. (New) The method of claim 55, wherein each prefix naming context is  
2 stored immediately subordinate to the repository naming context.

1 57. (New) The method of claim 55 further comprising the steps of:  
2 specifying an interface definition language file containing at least one  
3 interface definition language declaration;  
4 specifying a prefix naming context; and

5 storing each interface definition language declaration in the specified  
6 interface definition language file into the specified prefix naming  
7 context.

1 58. (New) The method of claim 57, wherein the step of storing each  
2 interface definition language declaration further comprises the steps of:  
3 creating an interface definition object for the interface definition language  
4 declaration;  
5 storing the interface definition object in the specified prefix naming  
6 context; and  
7 providing the interface definition object with a fully scoped object name  
8 including a prefix name from the prefix naming context in which  
9 the interface definition object is stored.

1 59. (New) A method of resolving name collisions by providing type  
2 support for multiple type definitions, comprising the steps of:  
3 defining in an interface repository a prefix naming context, the prefix  
4 naming context being adapted to resolve names subordinate to a  
5 repository naming context;  
6 storing the prefix naming context subordinate to the repository naming  
7 context in the interface repository, the prefix naming context  
8 forming an interface definition language root context for interface  
9 definition objects subordinate to the prefix naming context;  
10 specifying an interface definition language file containing at least one  
11 interface definition language declaration;  
12 specifying a prefix naming context;



13 storing each interface definition language declaration in the specified  
14 interface definition language file into the specified prefix naming  
15 context; and  
16 creating a data file identified as related to the specified interface definition  
17 language file, and containing an identification of the specified  
18 prefix naming context.

1 60. (New) A method of resolving name collisions by providing type  
2 support for multiple type definitions, comprising the step of:  
3 providing an interface repository including:  
4 a repository naming context; and  
5 a prefix naming context subordinate to the repository naming  
6 context, the prefix naming context serving as a root naming  
7 context for at least one interface definition language  
8 declaration, the prefix naming context being adapted to re-  
9 solve names subordinate to the repository naming context.

REMARKS

Claims 12-16 and 23-43 were pending in the application and have been canceled without prejudice. New claims 44-60 have been added to provide Applicants with claim coverage commensurate with the scope of the invention. Reconsideration and early allowance of claims 44-60 are respectfully requested.

Respectively submitted,  
Peter B. Kessler, et al.

Date:

May 24, 1999

By:



Kory D. Christensen  
Reg. No. 43,548  
Attorney for Applicants  
Fenwick & West LLP  
Two Palo Alto Square  
Palo Alto, CA 94306  
TEL: (650) 858-7657  
FAX: (650) 494-1417

# **METHOD AND SYSTEM FOR TYPE IDENTIFICATION FOR MULTIPLE OBJECT INTERFACES IN A DISTRIBUTED OBJECT ENVIRONMENT**

**INVENTORS:** Peter B. Kessler, Swee Boon Lim, Peter Vanderbilt, Michael  
5 L. Powell, Li-Wen Chen, Dwight F. Hare, and Alan Snyder

## **BACKGROUND**

### Field of Invention

This invention relates to the field of object oriented application and  
operating systems development, and more particularly, to methods and  
10 systems supporting the evolution of object interfaces in a distributed object  
programming environment.

### Background of Invention

Client-server computing is the predominant model for the  
management of computing resources, typically on a network of distributed  
15 computers. Servers are computers and application programs executing  
thereon that provide various functional operations and data upon request.  
Clients are computers and applications that request such services. While  
clients and servers may be distributed in various computers on a network,  
they may also reside in a single computer, with individual applications  
20 providing client or server functions, or both.

In a client-server system based on a object oriented development  
environment, a server implements and manages the functionality of a

collection of objects. An object consists of some state, and the interface of an object defines the methods, along with the parameters for the methods, to manipulate the state of the object. A client is code that invokes the methods of an object to manipulate the object.

5 In a distributed object programming environment, a client invokes an object through an object reference, without having information as to where the invoked object resides in the system, that is, without having specific information about the computer where the invoked object is stored or where it is running. Rather, the client passes the object reference through an object  
10 request broker (ORB) that locates the server, and passes back any output from the server to the client. To the user, the client and server, and any group of distributed objects, behave transparently as an integrated application, even though the objects may be distributed in various computer systems. One standard for a distributed object programming environment is the Common  
15 Object Request Broker Architecture (CORBA), specified by the Object Management Group (OMG).

In the CORBA environment, for the ORB to correctly handle requests, it must have access to the interface definitions of each of the objects it handles. The interface definitions can be made available to the ORB in one of  
20 two ways. First, the interface definitions can be directly incorporated into stub routines that represent object methods. Stubs implement proxy objects in the client. To invoke a server object, a client invokes a method on the proxy object for the server object. The stub routine of the proxy object

communicates with the ORB to transmit a request to the ORB. In this manner, a client can specify to the ORB the desired operation of the server object, since the interface of the server object is coded into the client. This process provides for static type checking of object definitions for clients  
5 written in programming languages such as C++. The client cannot access through its stubs, objects whose interfaces were not included in the stubs when the client was created.

Second, the interface definitions can be persistently stored in an interface repository, from which the ORB may dynamically access the  
10 interfaces definitions. With an interface repository, a client may invoke objects dynamically, without having information about the objects when the client is defined. When the client makes such an invocation, the client may access the interface repository to obtain the method signature of the invoked method. The client requires the method signature to construct a well-formed  
15 request to the object server and to receive the results of the operation. The ORB may also use the method signature to dynamically type check the parameters of the invocation. In the interface repository, the interface definition of an object comprises a description of the operations the object provides, along with the parameters of each operation, the exceptions it may  
20 raise, and additional context information. Each interface definition is itself stored as an interface definition object having operations that manipulate the underlying interface information. For example, an interface definition object

for the interface of a given server object will return to a client a description of the interface in response to a request.

As stated above, a client invokes an object by first obtaining an object reference. The client can invoke on the object reference a method to obtain  
5 another object reference to the interface definition object in the interface repository that handles the interface definition of the desired object. When the client has the interface of the object, it may invoke any of the methods or attributes defined in the interface.

Under the existing CORBA specification (OMG Object Request Broker  
10 2.0 Interface Repository RFP, OMG TC Document 94-11-7, available by anonymous ftp from omg.org), an interface repository must be coherent, meaning, in part, there is only one interface definition object in the interface repository for each type of object. As stated above, each object has an interface; the interface is described by a declaration in an interface definition language  
15 file, such as in the IDL defined by the OMG in TC Document 93-12-43. Each interface declaration has a scoped name. Two objects are of the same type if their interface declarations have the same scoped name. This is required to ensure the interface definition objects, which have the name given in the interface declarations, can be uniquely resolved. As the resolutions are based  
20 on type names, if there are two interface definition objects associated with a given type of object, name collision will result.

While this architecture ensures correct identification of objects with stable interfaces, it hinders the more realistic model of system

where object interfaces evolve over time. The evolution of interfaces is a normal aspect of most systems, and thus it is desirable

5 repository to support interface evolution.

*Must be kept  
↓  
Separate*

For example, in a desirable system installation there may be two co-existing environments, a deployment environment, and a development environment. In the deployment environment, objects and their interfaces are stable and unchanging. This is necessary to ensure that the behavior of each object is consistent, thoroughly tested, and known to all clients. In the development environment, however, the object interfaces are continually being improved and updated. These two environments must be kept separate so that client invocations in the deployment environment, where the client expects a particular interface for an object, do not invoke the development version of the object, which may not behave as expected by the client. However, when the development version of the object is stable, it is necessary to place it in the deployment environment for use by all clients.

The problem is that on a single system with a single interface repository, there is currently no easy way to separate the deployment and development environments so that multiple objects of the same type are represented by different interface definition objects. Thus, to differentiate development and development objects, the development objects must be distinct types from the deployment object, having their own interface

definition objects. Otherwise, if the interface of an object is updated, with methods added or deleted, existing clients holding an object reference would malfunction, since these clients would expect the previous interface of the object.

5           More importantly, in order to install a new object interface, it is most desirable to have both object interfaces for the current deployed and updated development objects operating at the same time, with some clients using the current interface, and some clients running the new interface. This allows a smooth transition between interfaces. However, an interface repository  
10           operating merely according to the OMG specifications does not allow multiple interface definition objects for a single object type. When requested to insert a second interface definition object where there already exists a first interface definition object for a given type of object, such an interface repository would either enter the second interface definition object and remove the first, or  
15           would reject the request as invalid. If the interface repository did allow the insertion of the second interface definition object, any later client request to resolve an interface by an object name would return an ambiguous result.

          In response to this problem the CORBA standard suggests the use of multiple interface repositories to insulate the development environment  
20           from the deployment environment. This approach severely limits the ability of the object developers to thoroughly test development objects with existing common resources, or existing client objects in the deployment environment. To test with existing clients, the deployed objects would have to be duplicated



in the development environment, resulting in greater requirements for system resources.

In addition, in order to install a new or updated object interface with separate interface repositories for development and deployment, the deployment interface repository must be shutdown, and the new interface installed, replacing the old one. This may impose unacceptable operating burdens on the system, and is not possible in large-scale systems that must run continuously, such as banking systems, or online transaction processing systems.

Accordingly, it is desirable to provide an improved system that allows a single interface repository to maintain multiple interface definition objects for a given object type, while providing correct interface identification and type checking for object requests.

### **SUMMARY OF THE INVENTION**

The present invention overcomes the foregoing limitations by providing fully scoped object names to objects, where the scoped names include a prefix that distinguishes objects in different naming contexts. In one aspect of the invention, a single interface repository includes a number of prefix naming contexts which are the IDL root contexts for different groups of interface definition objects. In each prefix naming context there can be interface definition objects for objects that have the same IDL name as objects in another prefix naming context. Name collision is avoided since each

interface definition object has a fully scoped name that includes the name of the prefix naming context, thereby resulting in different object types. In a preferred embodiment of this aspect of the invention, the prefix naming context is defined by a prefix object. In this embodiment, an interface repository loader loads IDL files containing IDL declarations into a specified prefix naming context, creating the prefix naming context if necessary. The interface repository loader further creates a new type of data file, an ifr file, that identifies the prefix naming context for each IDL declaration.

In another aspect of the invention, fully scoped object names are used in client stub and server skeleton code routines that access the object request broker environment. The fully scoped names here allow statically defined object types to be correctly identified and accessed by client objects. In a preferred embodiment of this aspect of the invention, an IDL compiler generates the stub and skeleton code routines.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

Figure 1 is a block diagram of a system for providing object type support for multiple interface definitions.

Figure 2 is an illustration of the interface repository of the present invention.

Figure 3 is an illustration of an instance of one interface repository conforming to the present invention.

[illegible]

## DETAILED DESCRIPTION OF THE INVENTION

### System Architecture

Referring now to Figure 1, there is shown a system for using the improved interface repository of the present invention. The system 100 includes a host computer 101 having a secondary storage 107 for long term storage of object implementation code and data, including servers and clients, and the like, an input device 109 and an output device 116 for receiving and outputting commands and data into the system 100, and an addressable memory 113 for storing server and client implementation code during execution by a processor 111. During execution by the processor 111, server 117 and client 105 objects exist as processes in the addressable memory 113. Additional server objects 117 and clients 105 may execute as a processes in other host computers 101 (not shown) connected along a network 103, or may execute as separate processes in any one of the host computers 101, or may execute in the same process. Each client 105 requests services or data from server objects 117 in host computers 101 on the network 103. The host computers 101 may be realized by most general purposes computers, such as a SPARCstation™ computer manufactured by Sun Microsystems, Inc. of Mountain View, Ca. Any other general purpose computer may also be adapted for use with the invention. The host computer 101 executes a general purpose operating system 114, such as Sun Microsystems' Solaris® operating system.

The system 100 incorporates an object request broker environment, satisfying the CORBA standards set by the Object Management Group in *The Common Object Request Broker: Architecture and Specification*, Rev. 1.2, OMG TC Document Number 93-12-43, available by anonymous ftp from omg.org. Other equivalent environments for distributed object systems may also be used. In the preferred embodiment, the object request broker environment is Sun Microsystems' Project DOE (Distributed Objects Everywhere). Accordingly, the system 100 includes an object request broker (ORB) 115, providing the functionality defined in the CORBA specification. Briefly, the ORB 115 handles requests for services from clients 105, determines the appropriate object implementation to service the request, conveys the request to the object implementation, and conveys the response to the request from the object implementation to the client 105. The ORB 115 allows the interface of the servicing object to be completely independent of the location or language of the object implementation. Each client 105 include stub code routines 137 for communicating with the ORB 115. Each server object 117 likewise has skeleton code routines 139 for communicating with the ORB 115.

There is also provided in system 100 an interface repository 119 in accordance with the present invention. The interface repository 119 provides the basic functionality of described in the CORBA specification, along with the further improvements described herein. An interface definition language compiler 135 is used to compile interface definition language files into the client stubs 137 and server skeleton 139 code routines for allowing objects to

directly access the ORB 115. The interface definition language files may be written in any suitable language, such as IDL as defined by the OMG. For ease of reference, interface definition languages referenced herein will be referred to as "IDL" but this is understood to include any interface definition language, and not merely IDL. An interface repository loader, IFR loader 143, takes a specified IDL file and loads the interface declarations contained therein into the interface repository 119. The operation of the IFR loader 143 is further described in the next section.

Referring now to Figure 2, there is shown in more detail one embodiment of the interface repository 119 of the present invention. In this embodiment, the interface repository 119 includes a number of naming contexts that define a hierarchy of information related to the interfaces of various objects in system 100. In the preferred embodiment, the naming contexts are instantiated with interface definition objects.

More particularly, there is provided a single repository object 121 that provides a number of operations for globally accessing the interface repository 119, including operations that locate a particular interface definition object by either an object name or repository identifier. Other interface definition objects are also provided, including interface objects 123, which define the constants, types, exceptions, operations, and attributes of each object in the system 100. Other interface definition objects are also provided, including module objects 125, exception objects 129, operation objects 141, and argument objects 143, which may be part of a naming context defined by an interface

object 123. Each of these objects may act as a container for other interface definition objects, thereby providing a hierarchical arrangement of naming contexts. Non-container interface definition object includes constant objects 127 and typedef objects 131. These interface definition objects are disposed as leaf nodes in the interface repository 119. The interface definition objects in the interface repository 119 are preferably derived from IDL files used to define the object interfaces. Together, the interface definition objects form a hierarchical name space that can be traversed to locate the interface of an object in response to a request by a client object.

To overcome the problems encountered when multiple interface definitions for a given type of object are accessed in a single interface repository, either through the name space of the interface repository 119, or through interface definition objects, the interface repository 119 and the interface definition objects therein have two additional features. First, the interface repository 119 includes at least one prefix object 133 as one of the available interface definition objects for defining the hierarchy of the name space. A prefix object 133 defines a naming context within the interface repository 119. In a typical embodiment there may be many such prefix naming contexts, each defined by a separate prefix object 133. Each prefix object 133 is an IDL root context for the interface definition objects subordinate to it. Each prefix object 133 represents a user chosen prefix that can be used to segregate various naming contexts. For example, two prefix objects 133 may be created to store distinct prefixes such as "deployment" and

"development," thereby segregating the deployment and development environments of the system 100. They may also be used to differentiate objects from different manufacturers, such as "SunSoft" or "DEC." Prefix naming contexts may also be used to differentiate different parts of computer systems, different company divisions, or other any type of differentiation useful to the users of the system 100. While references to the prefix object 133 and the prefix naming contexts in this document may refer to one or two such entities by way of example, it is to be understood that any number of prefix naming contexts and prefix objects 133 may be used in a single interface repository 119 according to the present invention.

Second, each interface definition object in the interface repository 119 has an object name that includes a simple name of the object, such as "foo," and each of the naming contexts in which it is placed, and including the naming context defined by a superior prefix object 133. Figure 3 illustrates one example of an interface repository 119 in accordance with the present invention, as might be used in a banking system. Here, there is the repository object 301, from which depend three prefix objects 133, a deployment object 303a, and a development object 303b, and a common object 303c. Under the deployment object 303a, there are additional interface definition objects, such as a bank module object 317a, an account object 305a, a debit object 307a, and a credit object 309a. Each of these objects 305a-309a are interface objects 123 because they define the interface of objects providing various services to clients. Accordingly, the account object 305a includes as part of its naming



context, further subordinate interface definition objects, such as an account balance object 311a, a withdrawal object 313a, and a deposit object 315a. Each of these interface definition objects define various operations supported by the interface of the account object 305a. Similarly, under the development  
5 object 303b, there are similar objects 305b-315b which represent the interface definitions which are currently being updated by the bank's developers to improve the performance of the respective objects, for example, by adding new operations, or deleting unnecessary ones.

The object names of the objects 305-317 include the full naming context  
10 path, including their superior prefix objects 133. Thus, in the deployment prefix name context under the deployment object 303a, the account balance object 311a has an object name "deployment/bank/account/balance", whereas the other account balance object 311b that is in the development prefix naming context under the development object 303b has an object name of  
15 "development/bank/account/balance." Thus, the use of the prefix naming objects 133 allows for unambiguous differentiation of multiple interface definition objects for a given object type.

In the preferred embodiment, the interface repository 119 of the present invention processes the full prefix of the object name when it  
20 receives a request for an object, and is thus able to resolve potential name collisions between two interface definition objects of a given object type. In this manner, a single interface repository 119 is able to maintain multiple interface definition objects corresponding to a single object type. This allows

clients of both interface definition objects to be present on the system 100 at one time. Thus, the interface of a given object can be modified and updated concurrently with, and in the same system as, the interface of its deployed version. This allows the development version to be tested with other  
5 existing client objects without duplication of system resources. It further eases the transition from a current version of an object interface to an improved one. With the prefix naming contexts, the improved interface repository 119 can support multiple client objects on both the current and updated interfaces. In addition, the transition or installation of an updated  
10 interface definition object does not require the system 100 to be shutdown.

### System Operations

In conjunction with the improved system architecture described above, the present invention provides improved operations for both the IFR loader 143 and the IDL compiler 135. These operations create and modify the  
15 interface repository 119 using the prefix objects 133, and generate client stubs 137 and server skeletons 139 that incorporate the naming contexts provided by the prefix objects 133.

### Load Interface Repository

The interface repository 119 cooperates with a facility to create and  
20 maintain interface definition objects. This facility is the IFR loader 143. Figure 4 is a flowchart illustrating the functional operation of the IFR loader 143 in loading the interface repository 119 according to the present invention.

The IFR loader 143 is invoked 401 to load a new IDL file, describing the interface of an object within system 100. The IFR loader 143 is passed arguments indicating the IDL file or files to be loaded, and the name of a prefix naming context in which interface definition objects resulting from the IDL file will be stored. For example, in one embodiment, a command line argument may be of the form **ifrload new\_object.idl -p deployment**, where 'new\_object.idl' is the file name of an IDL file to be loaded, and 'deployment' identifies a prefix object 133 defining a prefix naming context within which the interface definition is to be placed. This latter argument identifies where the IFR loader 143 is to subordinate the IDL root context.

The IFR loader 143 determines 403 if there exists a naming context defined by a prefix object 133 corresponding to the supplied prefix argument. If there is not already such a prefix object 133, the IFR loader 143 creates 405 a new prefix object 133 with the corresponding object name supplied in the prefix argument. This prefix object 133 is be placed 407 below the repository object 131, and becomes an IDL root context for the IDL definitions subordinated to it. For example, when the interface repository 119 shown in Figure 3 is initially loaded from an empty state with the command **ifrload bank.idl -p deployment**, the deployment object 303a would be created as a prefix object 133 subordinate to the root repository object, and the IDL definitions in bank.idl would be placed in deployment prefix naming context. If there is a prefix object 133 with the specified prefix, then the IFR loader 143 continues.

The IFR loader 143 reads 409 the specified IDL file, and obtains the IDL declarations contained therein. In many instances, in a given IDL file there will reference symbols or tokens that are defined in other files. For example, interface definitions of object operations may have return types that are  
5 defined in other IDL files. Or an IDL file may refer to various library functions, or the like for use in defining operations. These other IDL files will be referenced by an **include** statement in the given IDL file, or a similar construct in other interface definition languages. However, the objects types in the included files will be defined by names that are not fully scoped with  
10 the prefix bound names, but rather, with only the IDL names as used in the interface declaration.

For example, an IDL file `account.idl` may have object operations that refer to return types defined in a `datatypes.idl` file with an **#include** "`datatypes.idl`" statement:

```
15      // File: account.idl
      #include "datatypes.idl"
20      module bank{
          interface account {
              ...
25              datatypes::money balance(){
                  }
              ...
          }
30      };

      //File: datatypes.idl
      module datatypes {
35          interface money() {
```

```
...}  
}
```

5        Here, the return type of the balance operation is "money," which is defined in the included datatypes.idl file. Once installed in the interface repository 119, the interface definition objects for such operations must refer to interface definition objects of the return types that are defined in the interface repository 119. However, with the addition of the prefix objects 113,  
10    it is possible to have the interface "datatypes/money" defined in multiple prefix naming contexts. Any references to tokens that are defined in multiple naming contexts become ambiguous. Accordingly, the IFR loader 143 must be able to disambiguate the references to provide the definition of the tokens, such as the proper return type information. This is done in the preferred  
15    embodiment as follows.

      The IFR loader 143 determines 411 whether the IDL file contains references to other IDL files through the use of "include" statements. When an include statement is located, the IFR loader 143 obtains 413 the included file names. The IFR loader 143 reads the directory for the included file to  
20    determine 415 if a file exists with the file name of the IDL file followed by the suffix "ifr." The ifr file is a new type of data file that contains the name of prefix object 133 under which the interface definitions for the included IDL files were placed if they were previously loaded. For example, with respect to Figure 3, **ifrload -p common datatypes.idl** would load the datatypes.idl file

into a prefix naming context defined by prefix object 303c called "common."

This naming context includes information that is commonly available to both the development and deployment environments, such as the definition of data types. The IFR loader 143 will separately create a file named **datatypes.ifr** that contains the name of the prefix object, here **common**, under which the definitions in the data types file were placed. The ifr files, such as the **datatypes.ifr**, are stored in either the same directory as the included IDL file, or in a parallel file hierarchy. Those of ordinary skill in the art understand that the "ifr" suffix is merely used to differentiate the file from other types of files, and in itself is not significant. Other suitable file name schemes may be used to distinctly identify the file for the IFR loader 143.

If the ifr file for the included IDL file is available, the IFR loader 143 reads the ifr file, and obtains the naming context information stored therein. This allows the IFR loader 143 to resolve the reference, and obtain the correct information for object types, interface types, data types, exceptions, and the like. In this manner the existing interface definitions of common objects, or other existing objects, are available to any interface definition object in the interface repository 119, and thereby, a new interface definition object can be integrated in to the interface repository 119 without creating additional ambiguity. If the ifr file is not found, the IFR loader 143 may throw an exception and exit, or may perform any other desirable action, such as selecting a defined system-wide default prefix, and continue processing.

With the interface definitions from the IDL file and the included files, the IFR loader 143 creates 423 a naming context for each type declaration in the IDL file with the appropriate interface definition object. The naming contexts and interface definition objects are created according to the interface  
5 descriptions in the IDL files. For the example of the interface repository 119 of Figure 3, the IFR loader 143 will create from an IDL file describing a banking system consisting of a bank module object, with subordinate account, debit, and credit interface objects 123, each of which may have subordinate operation, constant, exception and other interface definition objects.

10 As each interface definition object is created it is placed 425 in the appropriate naming context, subordinate to the naming context of the specified prefix object. Each interface definition object is given 427 an object name that is fully scoped, including the name of the prefix object 113 under which it is subordinated. This allows the interface repository 119 to  
15 subsequently avoid name collision for interfaces whose simple object names, or IDL scoped names, are the same, but which reside in different prefix naming contexts.

Finally, the IFR loader 143 creates 429 a data file with a name format of `file_name.ifr` where `file_name` is the name of the IDL file that was loaded,  
20 and "ifr" (or any other suitable suffix) indicates the type of data type described herein. The ifr file contains the name of the prefix object 133 under which the interface definition objects were created.

## Generate New Stubs and Skeletons

The interface repository 119 is used to dynamically access object definitions. In addition, clients 105 may invoke server objects 117 using statically defined stub code routines 137 that provide access to the ORB 115.

- 5 To ensure that the stubs 137 correctly identify the object interfaces, the IDL compiler 135 provides the clients 105 with the prefix name scope in such stub routines 137.

In addition, the server object 117 or its proxy object in a client stub 137 must as be able to respond to a request from another client object 105 to  
10 identify the interface of the server object 117. For example, in the CORBA environment, this is done with an `_is_a()` request. The `_is_a()` determines whether an object is of a given type:

**CORBA::Boolean CORBA::Object::\_is\_a(object\_type)**

When there are two different interface definitions with the same IDL  
15 name, but which reside in different prefix naming contexts, then there is encoded in the stubs 137 and skeletons 139 the correct prefix contexts for each interface definition. More particularly, for each interface definition object with a unique fully scoped name, there is provided stubs 137 and skeletons 139 encoding the correct prefix naming context. When the stubs 137 and  
20 skeletons 139 are generated, the IDL compiler 135 provides the stubs and skeletons with an object name including a prefix for identifying the interface of the object. For example, in Figure 3, to generate stubs for the `datatypes.idl`



file where the interface is defined in the common naming context, the command `generate_stub -p common datatypes.idl` would be used in the Solaris® environment. This command results in each of the stubs 137 for the object type definitions having the fully scoped names, including the prefix  
5 name for the "common" prefix naming context. This process is similarly done when the skeletons are generated.

When the stubs 137 and skeletons 139 include the fully scoped names, the client stub code 137 will supply the correct interface to the server object 117 because the stub code 137 is statically typed, and uniquely identifies the  
10 correct interface. This information allows the `_is_a()` and any operation that specifies type to provide the correct interface definition of an object.

In conclusion, in any place or operation where there is a need to specify the IDL type, such as with dynamic invocation through the interface repository 119, or through statically compiled code in the stubs and skeletons,  
15 then the use of a prefix naming context provides a way to differentiate the multiple IDL types.

**WE CLAIM:**

1        1. A computer system providing type support for multiple type  
2 definitions, comprising:

3                an interface repository including:

4                        a repository naming context; and,

5                        a prefix naming context subordinate to the repository  
6 naming context, the prefix naming context serving as a root  
7 naming context for at least one interface definition language  
8 declaration.

1        2. The system of claim 1 wherein the prefix naming context further  
2 includes:

3                at least one naming context defined by an interface definition  
4 object and subordinate to the prefix naming context.

1        3. The computer system of claim 2 wherein at least one interface  
2 definition object has a fully scoped object name including a prefix name of the  
3 prefix naming context to which the interface definition object is subordinated.

1        4. The computer system of claim 1 wherein the prefix naming context is  
2 immediately subordinate to the repository naming context.

1        5. The computer system of claim 1 wherein the prefix naming context  
2 further includes:

3 at least one leaf node defined by an interface definition object.

1 6. The computer system of claim 1, wherein the prefix naming context is  
2 defined by a prefix object.

1 7. The computer system of claim 1, further comprising:

2 an interface repository loader that accepts as input parameters a  
3 specified interface definition language file containing at least one  
4 interface definition language declaration, and a specified prefix name,  
5 and installs the at least one interface definition language declaration in  
6 a prefix naming context having the prefix naming context in the  
7 interface repository.

1 8. The computer system of claim 7, wherein the interface repository  
2 loader creates a data file identified as related to the specified interface  
3 definition language file, and containing an identification of the specified  
4 prefix naming context.

1 9. The computer system of claim 7, wherein the interface repository  
2 loader creates the specified prefix naming context in the interface repository if  
3 the specified prefix naming context does not exist therein.

1 10. The computer system of claim 1, further comprising:

2 a memory device that stores the interface repository; and

3 a processing unit that executes operations of the interface  
4 repository loader.

1 11. The computer system of claim 10, wherein the processing unit further  
2 executes the interface repository loader to create a data file identified as related  
3 to the specified interface definition language file, and containing an  
4 identification of the specified prefix naming context.

1 12. A computer system providing type support for multiple type  
2 definitions, comprising:

3 at least one client object having a stub routine including a fully  
4 scoped name identifying a type providing an operation to the client  
5 object; and,

6 at least one server object having a skeleton routine including a  
7 fully scoped name identifying a type for the server object.

1 13. The computer system of claim 12, further comprising:

2 an interface definition language compiler that generates the stub  
3 routine in a client object.

1 14. The computer system of claim 12, further comprising:

2 an interface definition language compiler that generates a  
3 skeleton routine in a server object.

1 15. The computer system of claim 12, further comprising:

2 a memory device that stores the at least one client object, and the  
3 at least one server object; and,

4 a processing unit that executes a server object in response to an  
5 invocation of the server object by a client object.

1 16. The computer system of claim 15, wherein the processing unit executes  
2 an interface definition language compiler to generate the stub routine and the  
3 skeleton routine.

1 17. A method of providing type support for multiple type definitions,  
2 comprising the steps of:

3 defining in an interface repository a prefix naming context; and

4 storing the prefix naming context subordinate to a repository  
5 naming context in the interface repository, the prefix naming context  
6 forming an interface definition language root context for interface  
7 definition objects subordinate to the prefix naming context.

1 18. The method of claim 17, wherein each prefix naming context is stored  
2 immediately subordinate to the repository naming context.

1 19. The method of claim 17 further comprising the steps of:

2 specifying an interface definition language file containing at least  
3 one interface definition language declaration;

4 specifying a prefix naming context; and

5 storing each interface definition language declaration in the  
6 specified interface definition language file into the specified prefix  
7 naming context.

1 20. The method of claim 19, wherein the step of storing each interface  
2 definition language declaration further comprises the steps of:

3 creating an interface definition object for the interface definition  
4 language declaration;

5 storing the interface definition object in the specified prefix  
6 naming context; and

7 providing the interface definition object with a fully scoped  
8 object name including a prefix name from the prefix naming context in  
9 which the interface definition object is stored.

1 21. The method of claim 19, further comprising the step of:

2 creating a data file identified as related to the specified interface  
3 definition language file, and containing an identification of the  
4 specified prefix naming context.

1 22. A method of providing type support for multiple type definitions,  
2 comprising the step of:

3 providing an interface repository including:

4 a repository naming context; and

5 a prefix naming context subordinate to the repository  
6 naming context, the prefix naming context serving as a root  
7 naming context for at least one interface definition language  
8 declaration.

1 23. A method of providing type support for multiple type definitions,  
2 comprising the steps of

3 providing at least one client object having a stub routine  
4 including a fully scoped name for an object type providing an  
5 operation to the client object; and

6 providing at least one server object having a skeleton routine  
7 including a fully scoped name identifying an object type for the server  
8 object.

1 24. The method of claim 21, further comprising the step of:

2 providing a memory device that stores the at least one client  
3 object, and the at least one server object; and

4 providing a processing unit that executes a server object in  
5 response to an invocation of the server object by a client object.

## ABSTRACT OF THE DISCLOSURE

A system and method provides dynamic and static type support for multiple definitions of individual types in a distributed object environment. Fully scoped object names incorporating prefixes are used to distinguish objects. The fully scoped names are used for both dynamic type determination through an interface repository and for statically compiled types in client and server stub routines. In the interface repository, a prefix naming context is provided for each root interface definition language context, allowing multiple definitions for objects with the same interface definition language object name, where each such definition resides in a separate prefix naming context. In one embodiment, the prefix naming contexts are defined by prefix interface definition objects. In another embodiment, the fully scoped object names are embedded by an interface definition language compiler in stub and skeleton code routines.

RRS.



100

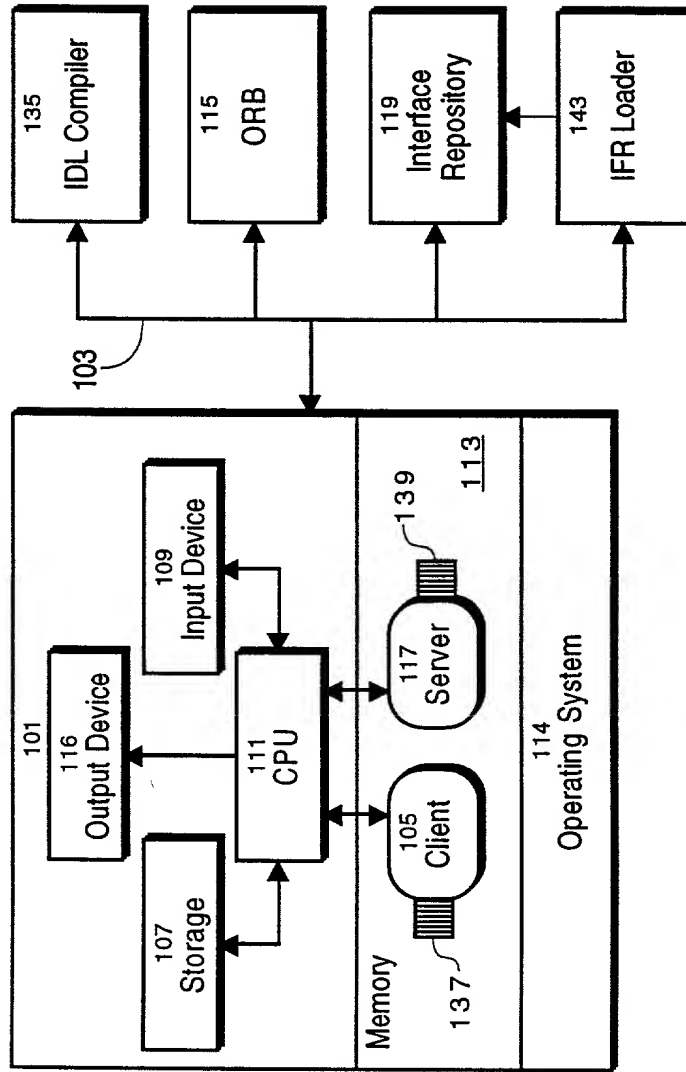


FIGURE 1

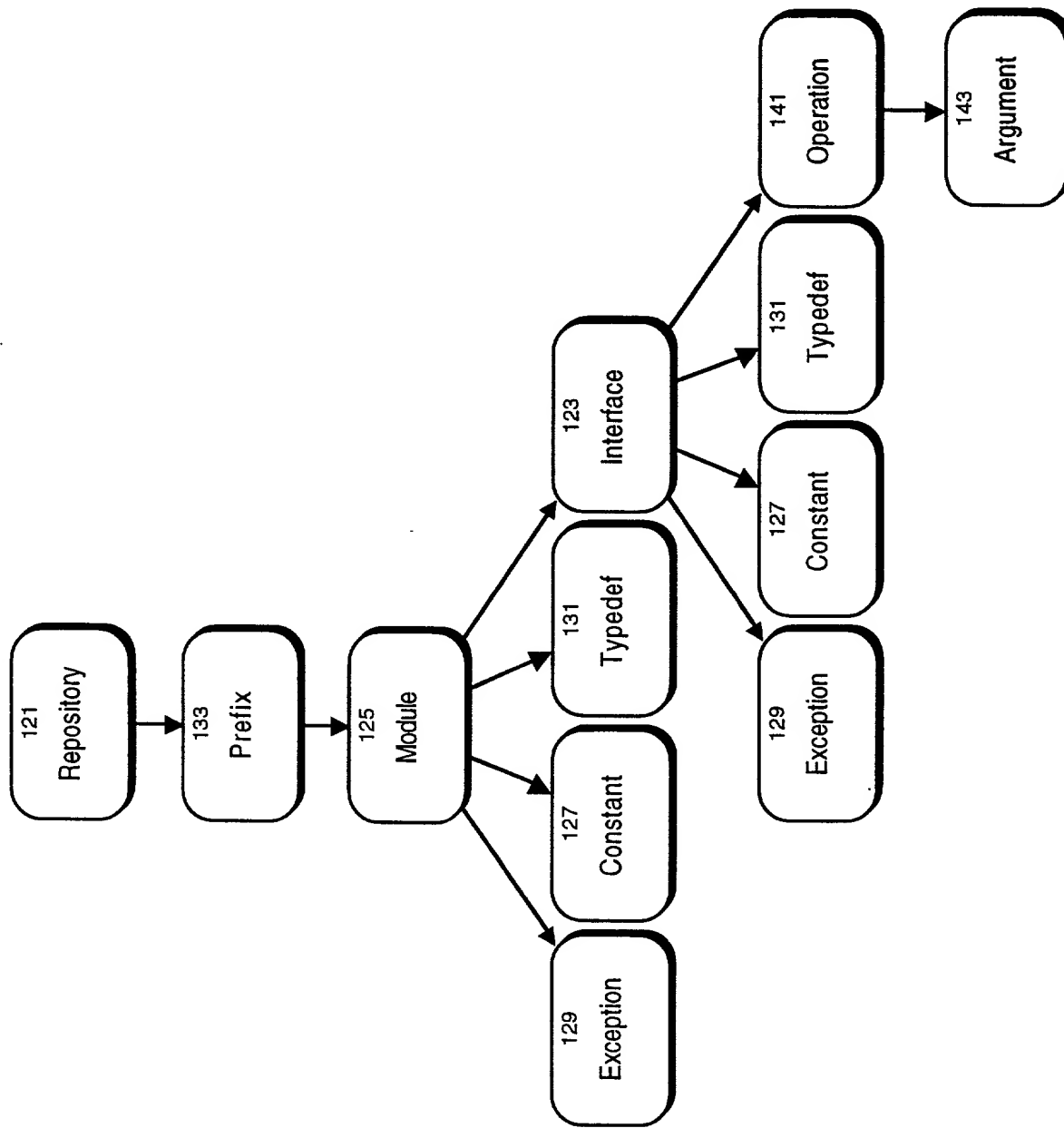


FIGURE 2

119

FIGURE 3

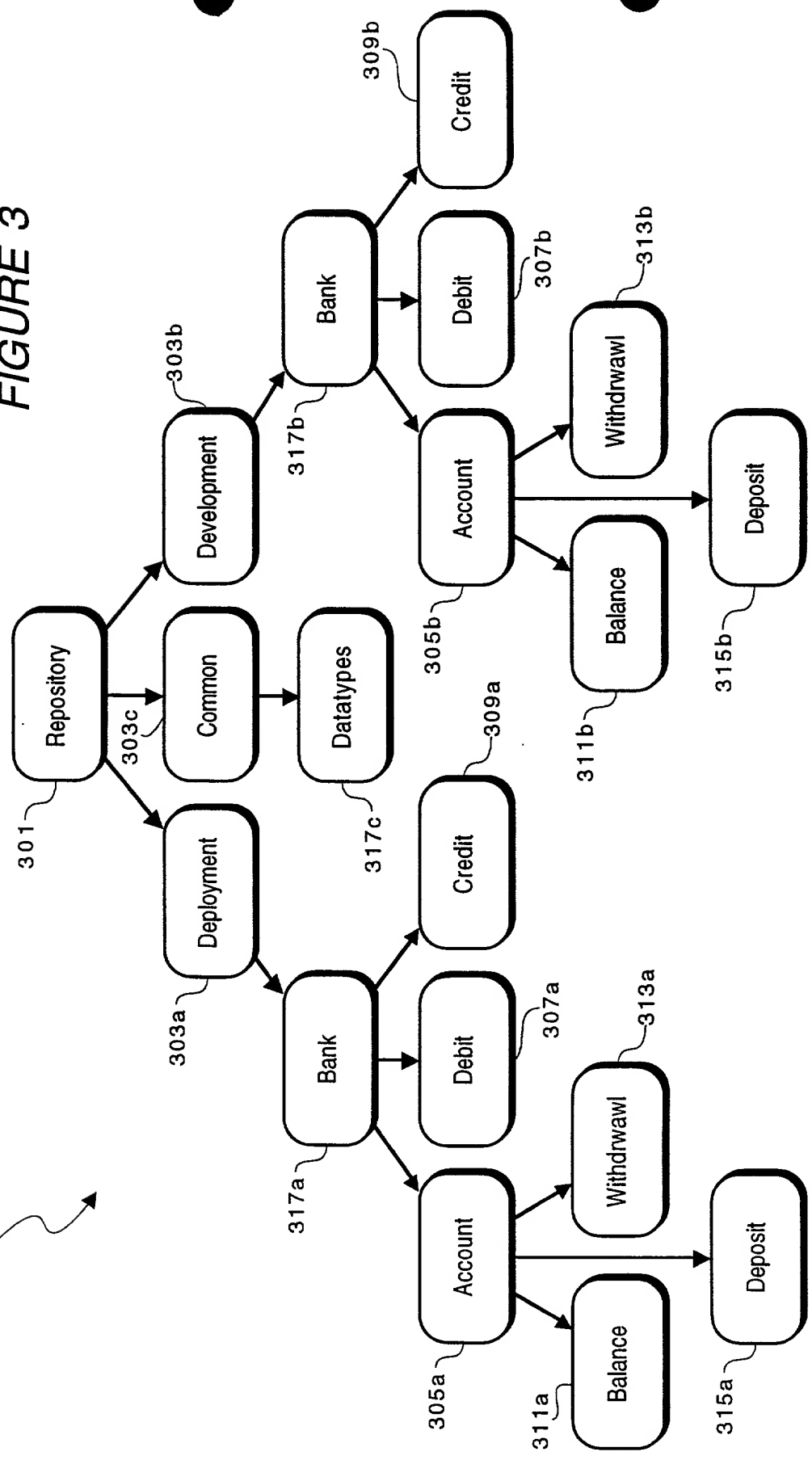
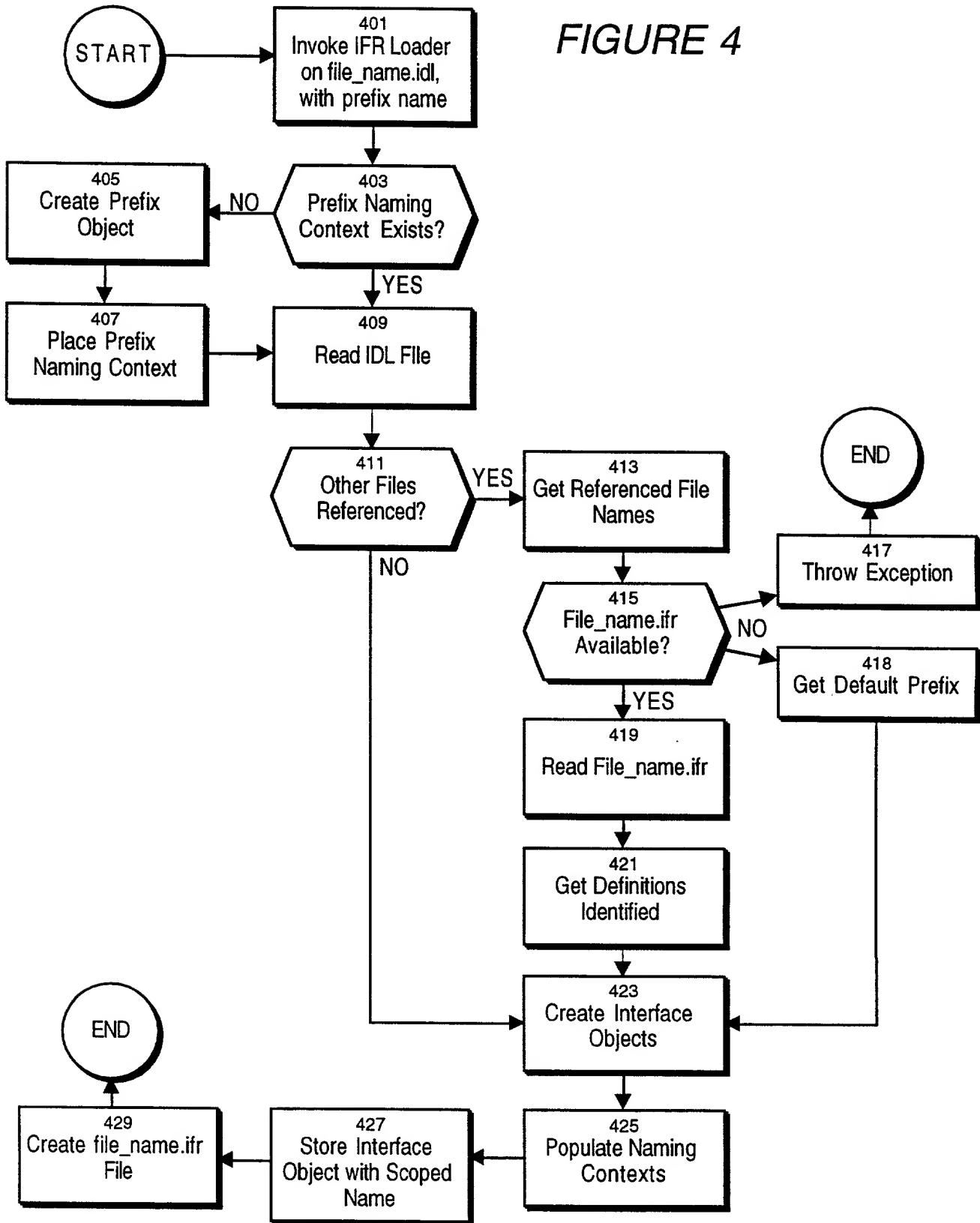


FIGURE 4



**DECLARATION AND POWER OF ATTORNEY  
FOR PATENT APPLICATION**

**ORIGINAL, CONTINUATION OR DIVISIONAL**

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

**METHOD AND SYSTEM FOR TYPE IDENTIFICATION FOR MULTIPLE OBJECT INTERFACES  
IN A DISTRIBUTED OBJECT ENVIRONMENT**

the specification of which (check one):

☒   
 ☐

is attached hereto.

was filed on:

Application Serial No.:

and was amended on:

(if applicable)

☐

was described and claimed in  
PCT International Application No.  
filed on:

and as amended under PCT

Article 19 on:

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, §1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) on which priority is claimed:

Prior Foreign or PCT Application(s): n/a

Priority Claimed?

Application Number	Country (or indicate if PCT)	Day/Month/Year Filed	<input type="checkbox"/> Yes	<input type="checkbox"/> No
			<input type="checkbox"/> Yes	<input type="checkbox"/> No

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) or PCT international application(s) designating the United States of America that is/are listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in that/those prior application(s) in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, §1.56(a) which occurred between the filing date of the prior application(s) and the national or PCT international filing date of this application:

Prior U.S. Application(s) or PCT International Application(s)  
Designating the U.S. for Benefit Under 35 USC §120: n/a

U.S. Application Serial No.	U.S. Filing Date	Patented	Pending	Abandoned	
U.S. Application Serial No.	U.S. Filing Date	Patented	Pending	Abandoned	
PCT Applications Designating the U.S.: n/a					
PCT Application No.	PCT Filing Date	U.S. Serial Nos. Assigned (if any)	Patented	Pending	Abandoned
PCT Application No.	PCT Filing Date	U.S. Serial Nos. Assigned (if any)	Patented	Pending	Abandoned

**POWER OF ATTORNEY:** As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

Judson D. Cary, Reg. No. 38,097  
Timothy H. Gens, Reg. No. 29,153  
Charles E. Gotlieb, Reg. No. 38,164  
David L. Hayes, Reg. No. 34,838  
Kenneth M. Kaslow, Reg. No. 32,246  
Matthew C. Rainey, Reg. No. 32,291  
Kang S. Lim, Reg. No. 37,491

Stuart P. Meyer, Reg. No. 33,426  
John T. McNelis, Reg. No. 37,186  
Leo V. Novakoski, Reg. No. 37,198  
Edward J. Radlo, Reg. No. 26,793  
Robert P. Sabath, Reg. No. 29,107  
James W. Rose, Reg. No. 34,239  
Timothy J. Crean, Reg. No. 37,116

Albert C. Smith, Reg. No. 20,355  
Phong K. Truong, Reg. No. 37,499  
J. P. Violette, Reg. No. 33,042  
Edward B. Weller, Reg. No. 37,468  
Lee Patch, Reg. No. 30,095  
Erwin J. Basinski, Reg. No. 34,773


**SEND CORRESPONDENCE TO:**

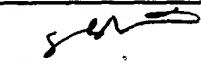
Albert C. Smith, Esq.  
FENWICK & WEST  
Two Palo Alto Square, Suite 600  
Palo Alto, CA 94306


**DIRECT TELEPHONE CALLS TO:**  
(name and telephone number)


Albert C. Smith, Esq.  
(415) 858-7110

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.


Full Name of Inventor: Peter B. Kessler  
Signature:  Dated: 3/21/95  
Residence: 4121 Vedosa Drive  
Palo Alto, CA 94306 Citizenship: U.S.A.  
Post Office Address: Same as residence


Full Name of Inventor: Swee Boon Lim  
Signature:  Dated: 3/21/95  
Residence: 405 Stierlin Road, #50  
Mountain View, CA 94043 Citizenship: Singapore  
Post Office Address: Same as residence


Full Name of Inventor: Peter Vanderbilt  
Signature:  Dated: 3/21/95  
Residence: 440 Beaume Court  
Mountain View, CA 94043 Citizenship: U.S.A.  
Post Office Address: Same as residence

Full Name of Inventor: Michael L. Powell  
Signature:  Dated: 3/21/95  
Residence: 2305 Emerson Street  
Palo Alto, CA 94301 Citizenship: U.S.A.  
Post Office Address: Same as residence

Case Docket No. 1725

Full Name of Inventor: Li-Wen Chen  
Signature:  Dated: 3/21/95  
Residence: 7725 Oak Meadow Court  
Cupertino, CA 95014 Citizenship: Republic of China (Taiwan)  
Post Office Address: Same as residence

Full Name of Inventor: Dwight F. Hare  
Signature:  Dated: 3/21/95  
Residence: 1 Breve Avenue  
La Selva Beach, CA 95076 Citizenship: U.S.A.  
Post Office Address: Same as residence

Full Name of Inventor: Alan Snyder  
Signature:  Dated: 3/21/95  
Residence: 4160 Briarwood Way  
Palo Alto, CA 94306 Citizenship: U.S.A.  
Post Office Address: Same as residence

Case Docket No. 1725